Классификация веб-страниц на основе алгоритмов машинного обучения

Борисова П. В.
Санкт-Петербургский Государственный Университет

Мышков П. С.
Санкт-Петербургский Государственный Университет

Незлобин А. А.
Stanford Business School

Петров А. Д.
University of Southern California

1. Аннотация

Долгое время появлявшиеся в литературе алгоритмы категоризации веб-страниц оставались в тени метода ключевых слов, который работал достаточно эффективно с англо-язычными сайтами. Поэтому возможности применения к этой задаче появившихся недавно алгоритмов классификации были недостаточно хорошо изучены [2,5,3]. Так, например, строковое ядро (String Subsequence Kernel, SSK) получило большее распространение в биоинформатике для классификации протеинов, нежели в веб-программировании для категоризации веб-страниц. Такие новые методы были непопулярны также из-за их несоответствия высоким требованиям к производительности , предъявляемых интернет-системам. Однако, при наличии должной оптимизации такие алгоритмы могут открыть новые возможности для создания простых в разработке категоризаторов, которые будут эффективны даже для языков со сложной морфологией и грамматикой. В данной работе приведён пример такого рода оптимизаций и предложено два классификатора, их реализующих. Результаты, полученные на практических тестах, очевидные возможности масштабирования, заложенные в эти алгоритмы – всё это даёт повод надеяться, что дальнейшее изучение этого вопроса окажется плодотворным.

Novel algorithms of web-page classification have been dominated by widely accepted keyword approach for a long time. The keyword approach has proved to be sufficiently effective for English web-pages. Therefore recently published classification algorithms have not been addressed in web-page classification research at an appropriate scale [2,5,3]. For instance, String Subsequence Kernel (SSK) received much larger attention in Bioinformatics for gene and protein classification than in web-programming for web-page categorization. Such novel methods have proved to be unpopular among Internet system providers also because of their high computational requirements. However, with application of certain optimization approaches, such algorithms can bring development of classification systems to a new level, where high efficiency can be achieved even for languages with complex morphology and grammar. This work represents an example of such optimization attempt and it provides two different realizations for such classifiers. Positive characteristics of presented results and scaling properties of these algorithms encourage further research in this area.

2. Введение и постановка задачи

В общем случае задача классификации заключается в нахождении правила, которое, основываясь на некоторых наблюдениях, относит каждый объект к одному из нескольких классов [4]. Для простоты будем считать, что существует всего два класса. Тогда формализовать это определение можно следующим образом:

Пусть задано пространство $\mathbf{X} = \mathbf{R}^N$ и множество классов $\mathbf{Y} = \{-1, +1\}$. Разделение точек в пространстве $\mathbf{X}$ на два класса задается неизвестным распределением $P(x,y)$. Тренировочное множество $\mathbf{T} = \{(x, y): x \in \mathbf{X}, y \in \mathbf{Y}\}$ состоит из пар, одинаково независимо распределенных (i.i.d) с распределением P. Другими словами, в $\mathbf{T}$ содержатся реализации (или примеры) случайной величины (x,y), или, проще говоря, представители каждого класса. Необходимо найти функцию f: $\mathbf{R}^N \to \{-1, +1\}$, правильно классифицирующую все остальные точки пространства $\mathbf{X}$ (напр., минимизирующую некоторую функцию стоимости).

Наилучшим решением считается функция f, минимизирующая средний риск классификации:

$$R[f] = \int l(f(x), y) dP(x, y),$$

где l – функция потерь, например $l = (f(x) - y)^2$.

К сожалению, минимизировать указанный интеграл напрямую невозможно, так как распределение P неизвестно. Поэтому приходится искать функцию f, которая близка к оптимальной в терминах имеющейся информации − тренировочного множества и свойств класса функций, из которого выбирается f.

Еще одна трудность заключается в том, что часто классифицируемые объекты изначально не являются точками в $\mathbf{R}^N$. В таких случаях возникает вторая задача – необходимо сначала сопоставить каждому объекту точку в $\mathbf{R}^N$, т.е. выделить признаки. Важно понимать, что от количества признаков и от их информативности зависит, насколько легко объекты разных классов будут отделимы друг от друга. Поэтому при классификации

текстовых документов эта проблема должна быть решена в первую очередь.

Самым распространенным классическим способом представления текстов является метод ключевых слов ([17]). Он используется в большинстве существующих систем, и многие исследования в области классификации также основаны на нем. Идея этого метода заключается в следующем. Для каждого класса текстов создается (вручную или автоматически) список характерных для него ключевых слов. Тогда каждый текст представляется вектором, компонентами которого являются частоты появления всех ключевых слов в данном тексте. У этого метода есть ряд неоспоримых преимуществ (например, высокая производительность), но есть и существенные недостатки:

- Полностью теряется информация о порядке слов, а она является важной семантической составляющей текста.
- Значение слова часто зависит от контекста.
- Со временем появляются новые слова, которые стоит считать ключевыми. Чтобы их учесть, надо перестраивать все пространство.
- Морфология языка затрудняет определение наличия слова в тексте.
- Зашумленность, вызванная грамотностью (а точнее, неграмотностью) пользователей Интернета, также затрудняет применение морфологических анализаторов и зачастую делает невозможным определение наличия слова в тексте.

Существуют различные модификации этого метода, частично решающие приведенные выше проблемы. Например, использование групп ключевых слов (или метод «ключевые слова в контексте») достаточно успешно решает вопрос о зависимости значения слова от контекста. Однако в целом перечисленные недостатки лежат в основе этого метода и полностью устранить их невозможно. Более того, последние две проблемы, связанные с морфологией и ошибками в письменной речи, в русском языке проявляются особенно ярко.

После выделения признаков обычно применяется один из стандартных алгоритмов решения задачи классификации, таких как нейронные сети (neural networks) и деревья решений (decision trees, [4]). У них всех есть одна общая черта – они пытаются отделить

точки одного класса от точек другого класса, явно оперируя их векторами в пространстве признаков $\mathbf{R}^n$ (или в преобразованном пространстве – например, в спрямляющем пространстве в скрытом слое нейронной сети). Однако все эти методы унаследуют недостатки, приобретённые на предыдущем этапе.

Поэтому в теории машинного обучения стали популярны подходы, альтернативные явному извлечению признаков – алгоритмы, связанные с использованием ядер [4]. Обозначим за $\mathbf{X}$ множество исходных объектов. Предположим, что существует отображение Φ, действующее из $\mathbf{X}$ в $\mathbf{F}$, где $\mathbf{F}$ – пространство, в котором образы исходных объектов линейно-отделимы (Φ – аналог функции выделения признаков). Тогда функция $k(x,y)$ будет ядром, если $(\Phi(x), \Phi(y)) = k(x,y)$. Таким образом, ядром называется функция, берущая два аргумента из исходного множества и вычисляющая скалярное произведение их образов в другом пространстве. Обычно, чтобы добиться линейной отделимости размерность $\mathbf{F}$ должна быть очень велика (в случае если $\mathbf{X}$ – линейное пространство, размерность $\mathbf{F}$ может оказаться значительно больше размерности $\mathbf{X}$), поэтому поиск разделяющей гиперплоскости в пространстве $\mathbf{F}$ может оказаться технически невыполнимой задачей. Тем не менее, если алгоритм классификации может быть записан только в терминах скалярного произведения, то возможно применение ядер и решение задачи в условиях ограниченных ресурсов. При этом отображение Φ, выделяющее признаки, напрямую не используется.

Известно несколько алгоритмов, которые могут быть адаптированы для применения с функциями ядер – это Метод Поддерживающих Векторов (Support Vector Machines, SVM, [2], [9], [11] и др.), Персептрон (Perceptron), Анализ Главных Компонент (Principal Component Analysis, PCA [4]), Метод Ближайшего Соседа (Nearest Neighbor) и другие. Все вместе они составляют класс алгоритмов, основанных на ядрах (Kernel Machines, [4]), и каждый лучше других приспособлен к задачам определенного типа в теории обучения, таким как кластеризация и регрессия. Для задач классификации наиболее успешно зарекомендовали себя Large Margin Classifiers (SVM [2,5,11], Perceptron [22]).

Суммируя, для решения задачи классификации необходимы три составляющие:

- Отображение Φ и пространство $\mathbf{F}$ (хотя бы неявные), в котором рассматриваемые классы линейно отделимы.
- Ядро $k(x,y)$, позволяющее неявно считать скалярное произведение $(\Phi(x), \Phi(y))$ в пространстве $\mathbf{F}$.
- Алгоритм разделения двух множеств гиперплоскостью, использующий только операцию скалярного произведения над элементами этих множеств.

В данной работе для классификации используются алгоритмы SVM [2,5,11] и Voted Perceptron [22] в сочетании с ядром SSK [3,8].

3. Идея исследования

Как отмечалось выше, строковые ядра не разбирают тексты на составляющие и тем самым позволяют избежать проблем, связанных с синтаксисом естественных языков и с зашумленностью веб-страниц. Однако вычислительная сложность строковых ядер очень высока, и не всегда их применение оправдано. Например, в английском языке синтаксис значительно проще русского, средняя длина слова и количество ошибок в письме меньше, и поэтому категоризаторы, основанные на ключевых словах, обладают достаточной эффективностью для практического применения. Этим объясняется маленькое число исследований в области оптимизации строковых ядер, теме потенциально богатой возможностями применения в разработке веб-приложений.

Использование строковых ядер позволяет надеяться на построение простого и эффективного классификатора, применимого ко всем языкам. В данной работе приведена попытка ускорить работу классификатора в следующих направлениях: понижение вычислительной сложности ядра, уменьшение количества вызовов ядра упрощением алгоритма классификатора и обеспечением возможности распараллеливания вычислений. В данном исследовании также приведена одна из первых попыток применения Voted Perceptron с SSK. Такое сочетание представляет интерес, потому что сравнительно высокая сложность SSK может быть компенсирована меньшим, чем в случае с SVM, количеством обращений к функции ядра. Такой классификатор будет обладать немного меньшей точностью, но зато может быть использован на объёмах данных, близких к производственным. Также голосующий персептрон легче приспосабливается к задачам многоклассовой классификации.

4. Методы и реализация

4.1 Метод опорных векторов (SVM)

Рассмотрим стандартную задачу классификации – пространство $\mathbf{R}^n$, два класса объектов – и предположим, что классы линейно отделимы. Уравнение разделяющей гиперплоскости имеет вид $(w,x)+b=0$, где w – нормаль к гиперплоскости. Полупространства, образуемые этой гиперплоскостью, задаются неравенствами $(w,x)+b>0$ и $(w,x)+b<0$. Т.е. для всех точек одного класса будет выполнено первое неравенство, а для точек другого класса – второе. Таким образом, мы ищем решающую функцию в виде

$$f(x)=Sign\ ((w,x)+b).$$

Ясно, что таких гиперплоскостей может существовать бесконечно много. Чтобы понять, какая именно нужна нам, вернемся к оценке риска классификации. В.Н. Вапником была доказана теорема ([2]), что риск классификации может быть оценен в терминах эмпирического риска (посчитанного на тренировочных данных) и слагаемого, учитывающего сложность класса функций, из которого выбирается решение:

$$R[f]\le R_{emp}[f]+\sqrt{\frac{h(\ln\frac{2n}{h}+1)-\ln(\frac{\delta}{4})}{n}},$$

где h – VC-размерность для рассматриваемого класса функций, неравенство выполняется с вероятностью $p>1-\delta$.

В [4] показано, что минимизация риска классификации эквивалентна решению следующей квадратичной задачи:

$$\sum_{i=1}^{n}a_i-\frac{1}{2}\sum_{i,j=1}^{n}a_ia_jy_iy_jk(x_i,x_j)\longrightarrow\max$$

при $0\le a_i\le C,\quad i=1,...,n$

и $\sum_{i=1}^{n} a_i y_i = 0$,

Учитывая количество индексируемых в Интернете страниц, размеры обучающей выборки следует оценивать как минимум несколькими сотнями тысяч. Это делает стандартные численные методы квадратичного программирования (метод Ньютона, прямодвойственные методы внутренней точки ([14]) и др.) технически неприменимыми.

Недавно было предложено несколько интересных алгоритмов ([6], [7]) решения этой проблемы. Одни опираются на факт, что решение задачи не изменится, если из матрицы выкинуть строки и столбцы, соответствующие нулевым множителям Лагранжа. Другие предлагают решать подзадачи фиксированного размера, двигаясь в направлении наибольшего спуска целевой функции. Для решения задачи оптимизации, возникающей при обучении SVM, наиболее эффективным оказался метод, известный как Sequential Minimal Optimization (SMO) ([7]).

Отличие SMO от других алгоритмов заключается в том, что на каждом шаге он решает минимально возможную подзадачу. Так как в задаче присутствует линейное ограничение, то минимальное количество множителей Лагранжа для совместной оптимизации равно двум. А для двух множителей задачу можно решить аналитически, не прибегая к численным методам квадратичного программирования. Несмотря на то, что приходится решать много подзадач, каждая из них решается настолько легко, что общее решение задачи может быть найдено очень быстро. К тому же, при таком подходе нет необходимости хранить даже часть Гессиана в памяти. Именно это алгоритм и был выбран для решения задачи оптимизации, возникающей при обучении SVM.

4.2 Многоклассовая классификация

Существует несколько подходов к решению задачи классификации с несколькими классами. Стандартный подход заключается в решении нескольких бинарных задач: последовательного отделения первого класса от остальных, второго класса от оставшихся и т.д., или выделения каждого класса из всего множества. После решения этих бинарных задач получается несколько обученных SVM, соответствующих каждому классу. Далее при определении класса нового объекта каждая SVM вернет

коэффициент принадлежности, и класс объекта будет определен по максимальному значению этого коэффициента.

4.3 Ядро String Subsequence Kernel (SSK)

Стандартный способ представления текстов – использование набора ключевых слов в качестве признаков – как отмечалось ранее, связан с множеством трудностей.

Использование ядер – это альтернатива явному извлечению признаков. Идея ядра SSK ([3]) заключается в том, чтобы сравнивать тексты по количеству общих подстрок, которые они содержат: чем больше общих подстрок содержат два текста, тем более похожими они считаются. При этом подстроки не обязаны входить в тексты неразрывно, и степень их разреженности по тексту определяет вес, с которым они входят в скалярное произведение.

Введем стандартные обозначения:

$\sum$ - входной алфавит,

$s \in \sum^n$ – строка длины n,

$|s|$ – длина строки s,

$s = s_1 \ldots s_{|s|}$,

$s[i..j] = s_i \ldots s_j$.

u – подстрока s, если существует такой набор индексов $\mathbf{i} = \{i_1, \ldots, i_{|u|}\}$, что $u = s_{i_1} \ldots s_{i_{|u|}}$. Длину подстроки u в s определим как $l(\mathbf{i}) = i_{|u|} - i_1 + 1$.

Теперь можно определить пространство признаков $\mathbf{F}$ и ядро k. Зафиксируем некоторое небольшое число n (например, 7). Размерностями (осями координат) в пространстве признаков будут всевозможные строчки длины n: $F = R^{|\Sigma|^n}$. Тогда если t – входной текст, то значением признака u будет сумма всевозможных вхождений строки u в текст t с учетом разреженности u по t:

$$\phi_u(t) = \sum_{i:u=t[i]} \lambda^{l(i)},$$

где $\lambda < 1$ – фактор, определяющий экспоненциальное уменьшение веса подстроки с увеличением ее длины в тексте.

Значения признаков показывают, насколько часто встречаются подстроки в тексте t, взвешенные в соответствии с их полной длиной в тексте. Скалярным произведением текстов s и t в $\mathbf{F}$ будет сумма значений признаков по всем строчкам длины n:

$$k_n(s,t) = \sum_{u \in \sum^n} (\phi_u(s) \cdot \phi_u(t) = \sum_{u \in \sum^n} \sum_{i:u=s[i]} \sum_{j:u=t[j]} \lambda^{l(i)+l(j)} .$$

Таким образом, мы получили функцию k: $\mathbf{X} \to \mathbf{R}_+$ со следующими свойствами:

- k сравнивает, насколько два текста похожи друг на друга,
- k обладает свойствами скалярного произведения (очевидно из определения k_n(s,t)) в пространстве $\mathbf{F}$.

Эта функция и называется ядром String Subsequence Kernel (SSK).

Несложно заметить, что в среднем тексты большей длины будут давать большее скалярное произведение. Полезно избавиться от этого эффекта, например, нормализовав ядро:

$$\widehat{k}_n(s,t) = \frac{k_n(s,t)}{\sqrt{k_n(s,s)k_n(t,t)}} .$$

Алгоритм динамического программирования [3] позволяет вычислить это ядро неявно за O(|n||s||t|) операций.

4.4 Аппроксимация и эффективное вычисление

Даже грубые тесты производительности показывают очевидную необходимость вычисления ядра со скоростью, на порядки превышающей O(|n|·|s|·|t|). Этого можно добиться, используя приближенное вычисление ядра.

Первое, что можно заметить в этом ядре – при подсчете оно позволяет строкам растягиваться по тексту сколь угодно сильно. Вспомнив, что вес вхождения строки уменьшается экспоненциально с ростом ее длины в тексте, становится понятно, что многие вхождения будут иметь почти нулевой вес. Эту особенность использует один из алгоритмов, основанный на суффиксных деревьях, называемых tries (от retrieval trees). Ограничив максимальную длину вхождения числом m, применением tries

можно добиться почти линейной сложности – $O(|n|\cdot|s+t|\cdot m^{m-n})$. К сожалению, оказалось, что и этот подход неприменим в нашем случае – для сравнения текстов желательно иметь n около 7 и m, хотя бы вдвое его превосходящим, и последний множитель (m^{m-n}) в этом случае будет слишком большим.

Подход, применённый в данной работе и показавший лучшие результаты, предполагает учитывание не всех строк, входящих в сумму в скалярном произведении. Он основывается на следующем утверждении:

Пусть $\mathbf{S} \subset \mathbf{X}$. Если $|\mathbf{S}| = \dim\mathbf{X}$ и вектора $\phi(s)$ ортогональны при $s \in \mathbf{S}$ (т.е. $k(s_i, s_j) = C\delta_{ij}$ для некоторой константы C), то верна формула:

$$k(x,y) = \frac{1}{C}\sum_{s_i \in S} k(x,s_i)k(y,s_i)\,.$$

Если теперь вместо полного $\mathbf{S}$ взять какое-то его подмножество $\mathbf{S}`$, ты мы получим приближенное равенство:

$$k(x,y) \cong \frac{1}{C}\sum_{s_i \in S'} k(x,s_i)k(y,s_i)\,.$$

Правильно выбрав $\mathbf{S}`$, можно добиться очень хорошего приближения для $k$. Эффективным способом выбора $\mathbf{S}`$ является следующий: рассмотреть все строчки длины n, непрерывно входящие во все тексты тренировочного множества, и выбрать из них наиболее часто встречающиеся m (число m определяет точность аппроксимации). По определению ядра все эти строки будут ортогональны. Естественно ожидать хорошего приближения, так как рассматриваются наиболее часто и наиболее непрерывно появляющиеся строки. Хотя это самый простой способ выбора $\mathbf{S}`$, не затрагивающий вопросы информативности признаков, он показывает очень хорошие результаты даже при сравнительно маленьком m.

Из определения $k`$ можно ожидать, что, по крайней мере, нижние ряды соответствующей ей матрицы будут заполнены очень мало. В случае аппроксимации, когда количество строк матрицы равно длине подстроки в определении ядра, этот эффект ещё более выражен – вся матрица заполнена не более чем на 10 процентов. Но

при динамическом программировании матрица всегда вычисляется полностью. В таких условиях эффективным решением становится применение рекурсивно-динамического программирования, то есть рекурсии с кэшированием.

Однако если непосредственно кэшировать рекурсивное вычисление по указанным формулам, то полученное решение будет в сотни раз медленнее динамического программирования. Это вызвано большим неявным перекрытием ядер в сумме для $k`$:

$$k'_i(sx,t) = \lambda k'_i(s,t) + \sum_{j:t_j=x} k'_{i-1}(s,t[1:j-1])\lambda^{|t|-j+2}, \quad i = 1,...,n-1$$

Чтобы решить эту проблему, введем дополнительную функцию $k```(s,t,l)$, учитывающую только подстроки, индекс последнего символа которых заканчивается после параметра l:

$$k'''_n(s,t,l) = \sum_{u\in\sum^n i:u=s[i]} \sum_{j:u=t[j],j_n>l} \lambda^{|s|+|t|-i_1-j_1+2} .$$

Очевидно, $k`(s,t) = k```(s,t,0)$. Рекурсивная запись для $k```$ выглядит как $k`$ с неполной суммой:

$$k'''_i(sx,t,l) = \lambda k'''_i(s,t,l) + \sum_{j:j>l,t_j=x} k'_{i-1}(s,t[1:j-1])\lambda^{|t|-j+2} \quad , \quad i = 1,...,n-1$$

.

Теперь, благодаря этой функции, можно выразить большее ядро $k`$ через меньшее:

$$k'_n(s,tv) = \lambda^{|v|} k'_n(s,t) + k'''_n(s,tv,|t|).$$

А значит, в сумме по строке t достаточно только один раз посчитать полное ядро $k`$ предыдущего уровня, все же остальные будут получены инкрементально с помощью $k```$. Осталось заметить, что полное ядро $k`$ также выражается через $k```$ по этой же формуле, если в качестве первого слагаемого взять ближайшее $k`$ из кэша. Если теперь занести все степени λ в массив, получается в среднем 10-кратное уменьшение количества умножений и соответствующий выигрыш в скорости. Дополнительное преимущество можно получить, отбросив вычисления с почти-нулевыми весами (λ^x при больших x). Также появляется возможность частично изменить

весовую функцию (λ^x), что в некоторых случаях может повысить показатели ядра.

Такой статистический подход к сравнению текстов сразу освобождает от использования морфологических анализаторов, так как допускает наличие шума. Более того, подстроки распространяются на несколько слов, что в некоторой степени позволяет учесть даже семантику текстов при сравнении. А при появлении новых текстов по теме нет необходимости перестраивать пространство признаков, которое можно назвать «всеобъемлющим» и размерность которого фиксирована.

4.5 Реализация

Описанные алгоритмы были реализованы на платформе Microsoft .NET 2 (beta 2), язык программирования – C#. Учитывая исследовательский характер работы, главными критериями при проектировании были открытость и расширяемость. Решение организовано в несколько библиотек с иерархическим пространством имен и открытыми интерфейсами:

Несмотря на то, что детальное рассмотрение преимуществ реализованной архитектуры выходит за рамки данного отчёта, стоит отметить, что структура программы предусматривает два места, в которых возможно параллельное вычисление:

- Многоклассовую задачу можно решать способом, при котором каждый класс отделяется от всех остальных. В этом случае каждая бинарная задача не зависит от других, и решать их можно параллельно на разных машинах. При последующей классификации вычисление коэффициентов принадлежности для каждого класса также будет производиться на разных машинах.
- В решении квадратичной задачи оптимизации, возникающей при обучении SVM, возможно разбиение Гессиана на матрицы большой размерности (> 2). Элементы этой матрицы – ядра с некоторыми коэффициентами – не зависят друг от друга, а значит, возможно их параллельное вычисление на разных машинах

4.6 Персептрон

Голосующий персептрон, подробно описанный в [22], обладает похожими на SVM теоретическими оценками

эффективности. Тестирование также показывает эквивалентную точность классификации, однако, различное время обучения и работы. Время обучения персептрона, в отличие от SVM, можно уменьшить за счёт незначительного уменьшения точности, однако SVM в малой степени выигрывает по времени классификации. Голосующий персептрон был использован в данной работе в совокупности с ядром SSK, чтобы получить оценки точности алгоритма в условиях близких к производственным. Стоит также отметить, что голосующий персептрон может быть достаточно легко и элегантно обобщён на задачу многоклассовой классификации [22].

4.7 Эксперименты и метрики эффективности

Для измерения качества классификации использовались 4 метрики: точность, правильность, полнота и мера F_1.

Точность – отношение количества правильно проклассифицированных текстов к их общему количеству. Эта метрика хорошо подходит для оценки качества решения бинарной задачи.

Другие три метрики удобно использовать в задаче выделения класса из множества. В этом случае классификатор определяет, принадлежит элемент классу (положительный ответ, P) или нет (отрицательный ответ, N). В обоих случаях ответ классификатора может оказаться как верным – True Positive (TP) и True Negative (TN) –, так и неверным – False Positive (FP) и False Negative (FN). Соотношения количеств ответов этих четырех типов и определяет указанные метрики:

- p (precision), правильность (часто именно эта метрика называется «точностью») – оценка вероятности того, что объект действительно принадлежит классу при условии, что классификатор так говорит: $p = \frac{TP}{TP + FP}$
- r (recall), полнота – оценка вероятности того, что классификатор определит принадлежность объекта классу при условии, что объект к классу действительно принадлежит: $r = \frac{TP}{TP + FN}$

- F_1 – гармоническое среднее правильности и полноты, взятое с весом 1: $F_1 = 2\frac{pr}{p+r}$

Наиболее распространённым способом поиска оптимальной настройки параметров классификатора является кросс-валидация. В данной работе параметров достаточно много, и в идеальном случае они должны оцениваться совместно. Такой полный тест не был проведён из-за высокой вычислительной сложности, ниже приведены настройки параметров, использованные в исследовании.

Настройки SVM:

- C = 1 (константа для соотношения расстояние-ошибки)
- $E_{kkt} = 0.01$ (точность вычислений при проверке условий Куна-Таккера)
- $E_{jo} = 0.0001$ (точность вычислений в задаче для двух множителей)

Параметры ядра (SSK):

- n = 5 (длина подстрок)
- $\lambda = 0.27$ (коэффициент веса подстроки)

Тестирование классификаторов проводилось на выборке из 19500 страниц, поровну выбранных из пяти рубрик: сотовые телефоны, футбол, психология, живопись, биология. Выборка была разделена поровну на обучающее подмножество и подмножество для тестирования. Предварительная обработка текстов проводилась с помощью нескольких простых эвристик, применение более сложных алгоритмов существенно на результатах не сказалось.

Адекватность использования именно 50-ти процентов выборки для обучения частично аргументируют следующие иллюстрации. Для проверки были выбраны 4 темы – спорт, политика, автомобили и страхование – и построена зависимость точности решения задачи дихотомии (разделение на два класса) от размера части выборки, используемой для обучения (таблица 1).

Размер тренировочного множества T, %	Точность			
	«спорт»-«политика»	«спорт-автомобили»	«автомобили-страхование»	Среднее значение
10	0.72	0.84	0.78	0.78
20	0.77	0.81	0.86	0.81
30	0.89	0.86	0.92	0.89
40	0.94	0.89	0.93	0.92
50	0.96	0.90	0.97	0.94
60	1.00	0.97	0.99	0.98
70	1.00	0.98	1.00	0.99
80	1.00	1.00	1.00	1.00
90	1.00	1.00	1.00	1.00

Таблица 1. Точность классификации при различных размерах тренировочной выборки.

Средние значения точности также проиллюстрированы на следующем графике 1:

Рисунок 1. Зависимость точности от размера обучающей выборки.

Из таблицы видно, что при размерах тренировочной выборки в 50% точность уже достигает 0.95, что ясно говорит о работоспособности метода в такой задаче. Также на графике легко прослеживается повышение качества классификации с увеличением тренировочного множества. Полное отсутствие ошибок в некоторых случаях (точность 1.00) объясняется небольшим количеством данных.

5. Результаты и выводы

Результаты тестирования классификаторов приведены в таблицах 2 и 3.

Рубрика	Точность		
	p	r	F1
Сотовые телефоны	0.86	0.57	0.69
Футбол	0.89	0.41	0.56
Психология	0.90	0.45	0.60
Живопись	0.65	0.41	0.50
Биология	0.42	0.67	0.52

Таблица 2. Результаты тестирования персептрона.

Рубрика	Точность		
	p	R	F1
Сотовые телефоны	0.89	0.62	0.73
Футбол	0.93	0.35	0.51
Психология	0.89	0.43	0.58
Живопись	0.72	0.38	0.50
Биология	0.46	0.72	0.56

Таблица 3. Результаты тестирования SVM.

Среднее значение F1 для персептрона составляет 0.57, а для SVM – 0.58, что показывает практическую идентичность этих алгоритмов на данном размере обучающего множества. Результаты, несведённые в таблицы, показывают, что при использовании большего количества данных для обучения SVM в среднем показывает более точные результаты, однако подобные улучшения являются несущественными для обоих алгоритмов. В целом результаты можно оценить как успешные, так как было почти

достигнуто значение F1 в 0.60, которое считается хорошим результатом для категоризатора сайтов на английском языке. Реализованный алгоритм допускает кластерное масштабирование и увеличение как точности, так и скорости вычислений за счёт привлечения дополнительных аппаратных ресурсов. В целом, мы надеемся, что полученные результаты привлекут дополнительное внимание исследователей к рассмотренным алгоритмам и способам их оптимизации.

6. Список литературы

[1] Вапник В.Н., Червоненкис А.Я. “Теория распознавания образов. Стохастические проблемы обучения.”, М.: Наука, 1974.

[2] V.Vapnik “An Overview of Statistical Learning Theory”, IEEE Transactions on Neural Networks, 1999.

[3] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, C. Watkins “Text Classification using String Kernels”, Journal of Machine Learning Research, 2, 2002.

[4] K.-R. Muller, S. Mika, G. Ratsch, K. Tsuda, Bernhard Scholkopf “An Introduction to Kernel-Based Learning Algorithms”, IEEE Transactions on Neural Networks, Vol. 12, NO. 2, 2001.

[5] T. Joachims “Text categorization with support vector machines: learning with many relevant features”, In Proceedings of ECML-98, 10th European Conference on Machine Learning, 137-142, 1998.

[6] T. Joachims “Making large-scale SVM learning practical”, Advances in Kernel Methods Support Vector Learning, MIT Press, 1999.

[7] J. C. Platt “Fast Training of Support Vector Machines using Sequential Minimal Optimization”, Advances in Kernel Methods Support Vector Learning, 1998.

[8] C. Carl “Kernels for Structures”, Publications of the Institute of Cognitive Science, Volume 9, 2004.

[9] C.J.C. Burges “A tutorial on support vector machines for pattern recognition”, Data Mining and Knowledge Discovery, 2(2).121-157, 1998.

[10] Arnulf B. A., Graf, Alexander, J. Smola, Silvio Borer “Classification in a Normalized Feature Space Using Support Vector Machines”, IEEE Transactions On Neural Networks, Vol. 14, No. 3, May 2003.

[11] N. Cristianini and J. Shawe-Taylor, “An introduction to Support Vector Machines”, Cambridge, 2000.

[12] N. Cancedda, E. Gaussier, C. Goutte, J.-M. Renders “Word-Sequence Kernels”, Journal of Machine Learning Research 3, 2003.

[13] C. Goutte, E. Gaussier “A Probabilistic Interpretation of Precision, Recall and F-score, with Implication for Evaluation”, ECIR’05, 2005.

[14] R. J. Vanderbei “Loqo: An Interior Point Code For Quadratic Programming”, Optimization Methods and Software, 451-484, 1999.

[15] C. Leslie, R. Kuang “Fast Kernels for Inexact String Matching”, Proceedings of COLT/Kernel Workshop, 2003.

[16] D. Hush, C. Scove “On the VC dimension of bounded margin Classifiers”, Journal on Machine Learning, 33-44, 2001.

[17] T. Joachims “A probabilistic analysis of the rocchio algorithm with TFIDF for text categorization.”, In Proc. of the ICML'97, 143-151, 1997.

[18] Т. Кормен, Ч. Лейзерсон, Р. Ривест “Алгоритмы: построение и анализ”, МЦНМО, 2000.

[19] В. Иванов, И. Некрестьянов, Н. Пантелеева “Расширение представления документов при поиске в Веб”, Труды RCDL2002, 2002.

[20] А.Б.Мерков “Основные методы, применяемые для распознавания рукописного текста”, доступно по адресу http://www.recognition.mccme.ru/pub/RecognitionLab.html/methods.html.

[21] N. Panteleeva “Using neighborhood information for automated categorization of Web pages”, Saint-Petersburg State University, available at http://meta.math.spbu.ru/~nadejda/papers/ista2003/ista2003.html.

[22] Yoav Freund, R. E. Schapire, “Large Margin Classification Using the Perceptron Algorithm. Machine Learning”, 37(3), 1999.